15-213: Introduction to Computer Systems
Written Assignment 3

This written homework covers how the control and procedures abstractions are implemented in machine level code. It also covers how non-basic data types (arrays and structs) are stored in memory.

Directions

Complete the question(s) on the following pages with single paragraph answers. These questions are not meant to be particularly long! Once you are done, submit this assignment on Canvas.

Below is an example question and answer.

Q: Although arrays can implicitly decay to pointer types in C, and operations such as indexing and pointer arithmetic work on both types, arrays are not pointers. With this being said, show an instance when an array behaves differently than a pointer in C. Additionally, show an instance where an array's implicit decay to a pointer type is useful.

A: One instance where arrays behave differently than pointers in C is with value assignment. Pointer variables can be assigned different values, but the addresses of statically declared arrays cannot. In other words, it is possible to treat a pointer variable as an array (i.e. interpreting a pointer's value as a dynamically allocated array), but it is not possible to treat a statically declared array as a pointer variable. One instance where the implicit decay is useful is when you pass an array as an argument. When arrays are passed as arguments to functions, they are typically converted to pointers.

Grading

Each assignment will be graded in two parts:

1. Does this work indicate any effort? (e.g. it's not copied from a homework for another class or from the book)
2. Three peers will provide short, constructive feedback.

Due Date

This assignment is due on September 30th, 11:59 PM EST. Remember to convert this time to the timezone you currently reside in.

Question 1

1. Wow Abi found a coolFunction! but unfortunately she can't find the code for the function **mystery**. All she has is this assembly snippet. What value should Abi input if she wants to print out "left shark"? Please explain briefly why your answer works.
2. How does this assembly safely use the callee-save register, %rbx?

```
int coolFunction(int n){
        if ( mystery(n)){
                printf( "left shark\n");
         }else{
                 printf( "boo\n");
        }
        return 0;
}
```

```
doSomething:
0x0000000000400529:   sub       $0x10, %rsp
0x000000000040052a:   mov       %rbx, 8(%rsp)
0x000000000040052c:   mov       %edi, %ebx
0x000000000040052d:   test      %ebx, %ebx
0x000000000040052f:   jle       40053d <doSomething+0xf>
0x0000000000400531:   mov       %ebx, %edx
0x0000000000400533:   sub       $0x1, %edx
0x0000000000400536:   jne       400533 <doSomething+0x6>
0x0000000000400538:   lea       (%rbx, %rbx,1),%eax
0x000000000040053a:   mov       8(%rsp), %rbx
0x000000000040053b:   add       $0x10, %rsp
0x000000000040053c:   retq
0x000000000040053d:   mov       $0x0, %eax
0x000000000040053e:   mov       8(%rsp), %rbx
0x000000000040053f:   add       $0x10, %rsp
0x0000000000400541:   retq

mystery:
0x0000000000400542:   callq     400529 <doSomething>
0x0000000000400547:   cmp       $0x1aa, %eax
0x000000000040055c:   sete      $al
0x000000000040055e:   movzbl    $al, %eax
0x0000000000400560:   retq
```

Please write your answer to Question 1 here.

Question 2

We learned in lecture that a struct may include padding at the end. Padding at the end of a struct does not change the alignment of any fields in the struct; why is it necessary for a struct to be aligned?

Fill in the data of this struct in memory. Let each block represent one byte. Fill in a block with a field's name to indicate that the field occupies that byte. Use X's to represent padding.

How can we index into a[2] of the struct given that the struct begins at address r?

```
typedef struct babyShark{
        char c
        int a[3]
        long b
        char *d
        char e

} babyShark_t;
```

babyShark_t:
